Real-time Communication Using WebRTC

Haytham Ali
Georgia institute of technology
Atlanta, Georgia USA
Haytham.Ali@gatech.edu
webhay@hotmail.com

ABSTRACT
This paper introduces a new Real-time Communication technology. Some of the technology's applications could be in the education field, but are not limited only to education. The technology could be used with any system need, video and audio conferencing, such as medical uses or even online gaming.

WebRTC is a free and open source project that provides web browsers and mobile applications with real-time communication (RTC) via a simple application programming interface (API). It allows video, audio and data to work inside a web page by allowing a direct peer to peer communication between two browsers, eliminating the need for a server or browser plugins or downloading a native application.

The WebRTC project is supported by Google, Microsoft, Mozilla and Opera, and it is being standardized through the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF). (Uberti, Justin; Thatcher, Peter, 2018).

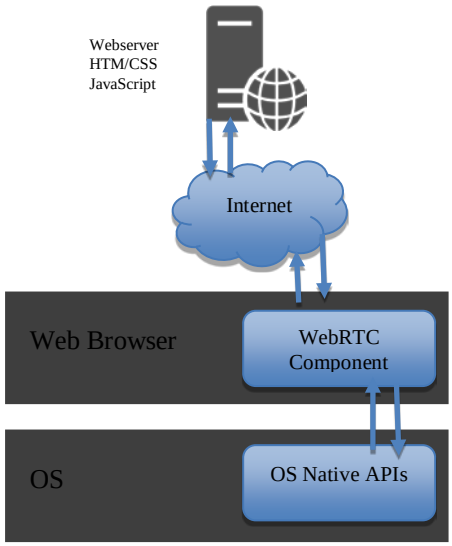
Author Keywords
WebRTC; Real-Time; Communication; Education Technology; live video conferencing.

INTRODUCTION
Web Real-time Communication (RTC) is a new feature added to web browsers. These features will enable the web browsers to directly communicate in real-time fashion.

Browsers, in general, do not communicate directly. The old model was the browser communicating with a server, usually web server, and then the server keeps the data until another user interacts with this browser to get the data.

WebRTC Introduction:
The browser model without WebRTC you may already know. It can decode or translate the HTML and JavaScript tags on a web page.

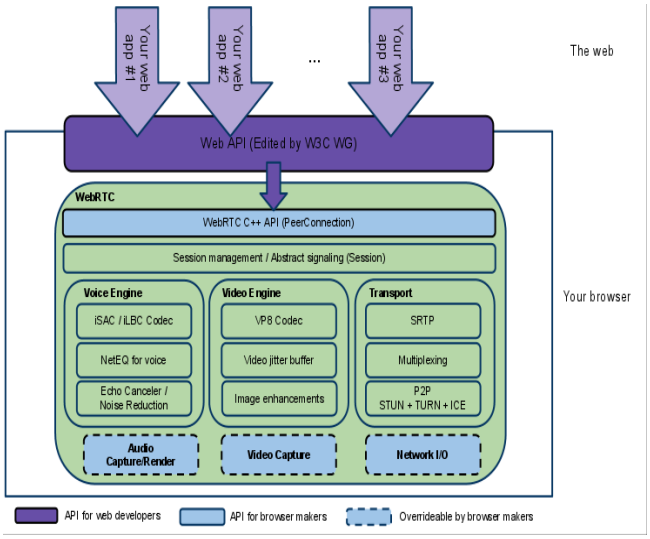
This translation is possible because all the browsers added components to comply with W3C HTML tags standards. The same design, when it comes to WebRTC browsers model, each browser must add another component to be able to support WebRTC. Most of the major browsers added the RTC (Real-time Communication) component since 2014 as a part of each browser release. Below is a simple diagram showing the RTC component.



From the above diagram we can clearly see the WebRTC component embedded in the browser. It takes care of calling the native OS audio and video APIs.

JavaScript plays an important role in WebRTC, since the technology runs on the browser. JavaScript is the perfect choice as the programming language to mediate between the WebRTC and the application through a set of exposed APIs.

WEBRTC ARCHITECTURE



The browser hides the big part of the WebRTC implementation, all the work of dealing with capturing the media, session description, and network transmissions, it is all abstracted away from the surface, and only exposing simple APIs.

The application can only call the Web API which is standardized by W3C for the Web developers to consume (Uberti & Dutton, n.d.).

IMPORTANT COMPONENTS IN WEBRTC

MEDIA STREAMS

Devices today come with multiple media hardware capabilities. In mobile devices you get front camera, back camera and audio microphone. In desktop computers and tablets you get the same set of media sources. Each of these media sources can generate a media stream.

This media stream can be captured by WebRTC Browser API and the API will deliver this stream to the JavaScript API as a variable for the application developers to display on a page like any normal video feed.

PEER TO PEER

One of the distinctive WebRTC advantages is the fact it is a Peer to Peer communication. This obviously means the video will flow directly from a user's computer to another.

SESSION DESCRIPTION

In general, for any communication to happen between any two devices, there is some handshake mechanism that must be done first, to establish the session between the two devices and then the flow of the messages or packets can start.

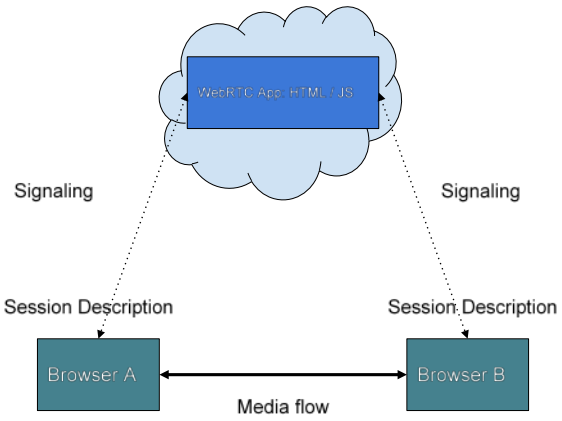
It's the same thing with WebRTC. There is a SDP (Session Description Protocol). It is a set of messages that should be exchanged between the two browsers before they start to send media streams.

To exchange the Session Description data, we need an intermediate machine. This is where the signaling concept comes into the picture (Alan, 2014).

SIGNALING

Let's say Browser A is trying to communicate with Browser B. Browser A will send its Session Description data to the Signaling Server S, and the same process for Browser B too.

Once this process is done, each browser now knows everything needed to establish a connection and send the media stream to the other browser. The below diagram will help explain this concept (Alan, 2014).



THE ROLE OF SIGNALING

Signaling has an important role in real-time communications:

1- Negotiation of media capabilities and settings between the devices in the same call
2- Identification and authentication of participants in a session
3- Controlling the media session, indicating progress, changing and terminating the session
4- Glare resolution, when both sides of a session try to establish or change a session at the same time

WHY SIGNALING HAS NOT BEEN STANDARDIZED

Signaling is not standardized in WebRTC to allow interoperability between different browsers. Signaling is a matter between the web browsers and the web server, not between two browsers in the same call or session. In that case the server selects the signaling protocol and ensures that users of web applications or site support use the same protocol (Alan, 2014).

SIGNALING AND MEDIA NEGOTIATION

The most important function of signaling is the exchange of information contained in the session description protocol (SDP) object between the browsers. SDP contains all the

information necessary for the RTC to work, including the types of media (audio, video and data) codecs used (Opus, G.711, etc.) and information about the connection bandwidth. One other role for signaling is ICE (Interactive Connectivity Establishment).

This is the candidate address representing the IP address and UDP ports where potential media packets could be received by the browser. But it cannot be started until the candidate address has been exchanged over the signaling channel.

IDENTIFICATION AND AUTHENTICATION

The signaling channel can also provide the identity for the participants in a call. For instance, in a web application the application will take care of authenticating the users first, and when a user needs to call another user, the web application will present the username for signaling channel as an authenticated ID.

WebRTC has other ways to authenticate the users without relying on external applications. By using the media channel, which does not rely on trusting external applications, a caller identity and authentication can be provided in the media path without relying on the signaling channel at all.

CONTROLLING THE MEDIA SESSION

Signaling is required to initiate the call; however, it is not required to indicate status or to terminate a session. There is ICE state machine in the browser which can provide this information.

GLARE RESOLUTION

Glare resolution is when both sides of the communication are trying to change the session at the same time. It is a race condition that could result in a nondeterministic state for the session. The SIP signaling protocol has built in glare resolution.

SIGNALING TRANSPORT

WebRTC requires a bi-directional signaling channel between the browsers. There are transport protocols that are commonly used for WebRTC signaling: HTTP, Web Socket and the data channel.

HTTP TRANSPORT

HTTP can be used as a signaling transport and the browser can just use the HTTP mechanism of sending data to the server by using the GET or POST method to send and receive signaling information. One important thing about HTTP is the server, which is used for signaling in a different IP or domain name. The server must support CORS to all the requests coming from different machines. HTTP protocol can also be secured by using HTTPS version and it also allows authentication.

WEB SOCKET TRANSPORT

Web Socket can be used as signaling protocol as well. The connection starts as a HTTP connection and then turns to web socket connection later. Also, if the server allows CORS, the Web Socket server can be at a different IP address than the web server. As in the below figure it is very similar to HTTP signaling (Alan, 2014).

DATA CHANNEL TRANSPORT

The data channel can't be used only as a signaling mechanism for WebRTC communication. The application has to rely on any other signaling protocol first and once the data channel is established between peers, all the video and audio communication can use it for signaling (Alan, 2014).

SIGNALING PROTOCOL

There are a few protocols that could be used as a signaling protocol. The choice of signaling protocol is an important one, including the custom signaling protocol implementation. We are not going to explain all of them, just the most used ones (Alan, 2014).

WEB SOCKET PROXY

Web Socket protocol can be used as a signaling protocol. Each browser opens an independent web socket connection with the same server, and the server bridges the connections. When information is received from a Web Socket, it is broadcast to all open Web Sockets connections (Alan, 2014).

SIP OVER WEBSOCKETS

SIP over Web Sockets is another protocol that can be used for signaling. SIP is a key protocol used to replace PSTN. SIP is also used in enterprise communication systems. It provides instant messaging (IM) and presence. SIP also can use UDP, TCP, SCTP or TLS as transport (Alan, 2014).

JINGLE OVER WEB SOCKETS

Jingle is a part of XMPP (Extensible Messaging and Presence Protocol), aka Jabber. Jingle converts SDP sessions to an XML format, which can then be transported over TCP or TLS. This protocol is used by Google Talk and other enterprise IM services. XMPP clients written in JavaScript would be downloaded from a web page and run (Alan, 2014).

DATA CHANNEL PROPRIETARY SIGNALING

The data and format sent over the data channel can be completely proprietary. Some approaches do not even send SDP objects over the data channel. Instead, using custom messages and then locally, it can be used to generate SDP (Alan, 2014).

DATA CHANNEL USING AN OVERLAY

An overlay network is a network which overlays or sits on top of another network. The overlay hides the underlying topology and architecture and provides an alternative way to address and message other members of the overlay network. There is some technology used to implement and utilize an overlay as a distributed Hash Table. The data channel can be used to establish an overlay signaling network for WebRTC.

THE WEBRTC APIS

WebRTC has three major features:

1- Acquiring audio and video
2- Communicating audio and video
3- Communicating arbitrary data

And there are three major JavaScript APIs exposed to call these APIs:

1- Media Stream (aka getUserMedia)
2- RTC Peer Connection
3- RTC Data Channel

MEDIA STREAM

We refer to the cameras and microphones in user computers as Local Media. Therefore, the first step is to obtain a local media stream. That can be done in multiple ways. Here is an easy way to do that.

Just one JavaScript method we can call:

getUserMedia() This method will return a callback method with a single media stream. For privacy issues the browser must get the user's permission to access the user's camera and microphone. The browser will prompt the user to allow access. Once the access is granted the application will receive a stream from the camera and the microphone (Alan, 2014).

RTC PEER CONNECTION

It is important to note this connection is a "peer" connection. That means it can be done with any computer or device on the Internet. And the connection is directly established between the two machines with no server involved.

The video/audio packets will flow from one browser to another. To establish a peer connection we just need to call this method **RTCPeerConnection()** (Alan, 2014).

One very important fact worth mentioning is how many connections will be created for each user added to a conference call.

Let's say, for example, a call started with two users, such as user A and user B. That means one peer connection will be created between the two users' browsers, a connection from A to B.

If another user joins the call, say user C, this will change the connections per each user in the call. As we explained earlier, this will be a peer connection which means each user must be connected to each user's machine in the call.

To understand how many peer connections will be created per machine:

Connections per machine = number of users in call - 1

Connections per machine = 3 – 1

Each machine will have 2 peer connections established if the call has 3 users.

How about 50 users in the call? That will create 49 peer connections per machine in the call, which is not a scalable design. Later in the paper, the STUN and TURN servers will be introduced to overcome this problem (Alan, 2014).

RTC DATA CHANNEL

Data channels support high volume and low latency connections; a data channel is a non-media channel that supports data transfer only. It bypasses servers and provides the web developers with configurable channels to transfer data.

Data channels in WebRTC are built on Web Sockets. In this way it gets a real-time feature. Simple methods like send on message handler can be set to get the data. Data channel also uses the same peer connection as the media, which is a good thing, because that means that only one offer/answer negotiation process is needed (Alan, 2014).

Data channels also support the two modes of delivery: guaranteed (reliable) and fast (unreliable) delivery. The former can be used for critical events and the latter can be used for game position updates (Alan, 2014).

WEBRTC AND NAT

Today's network designs have NAT enabled already on them. Most of the devices support NAT and enable it once you use them. Earlier, when the Internet started, each device was connected to the Internet got assigned a public IP address, which anyone else connected to the Internet could ping or even communicate to this device.

This setup wasn't secure and the IP addresses were kind of running out, because the connected devices increased dramatically fast. NAT came into play to resolve this issue. By just assigning the router (which multiple devices and computers are connected to) with only one public IP address, the router will give the connected devices, say a virtual IP address or private IP address.

The private IP address is not reachable from outside of this network. But it is reachable within the network. For a computer or device in this private network to connect to the outside world, such as streaming a video from YouTube, the router must translate the device's private address to its public address, and translate back from public address to private address and then forward the packets to the connected device in the private network. For instance, if you are watching a YouTube video, this process will happen with each small packet delivered to the network's main router.

What is the WebRTC problem then?

Peer to Peer means the two computers have public and reachable addresses. From what was explained earlier about NAT, this problem can't happen. The solution was to introduce STUN and TURN servers (Alan, 2014).

STUN SERVER

This is a very simple server setup. STUN stands for Session Traversal Utilities for NAT. The idea is to allow the browser (computer) to find out its public IP address, which represents the network that this computer is a part of. This will be, in most cases, the public IP address for the main router or NAT device. The same process happens with the other peer

browser, too. And both browsers share their network public IP address to finalize this handshake (Alan, 2014).

TURN SERVER

TURN server has the same purpose as STUN server which is a workaround NAT issue. However, this time it's a little different and it handles a different scenario. TURN stands for Traversal Using Relay around NAT. The browsers (computers) query a TURN server to obtain not the public IP address of the network, but media relay server address, which is a public IP address for a media server used to receive the actual media packets, such as audio/video, for instance, and relay it to the other peers in the call. This scenario is needed if the direct peer to peer communication is not allowed in the network. This is still not a very common scenario. Around 15 to 25 percent of the calls end up using TURN servers (Alan, 2014).

SECURITY

WebRTC uses the same available network security protocols features. It can use signaling over HTTPS, STUN and TURN server support using multiple authentication mechanisms. Token based authentication is the most used, in which each WebRTC application can get authenticated by receiving a token from the STUN and TURN server. The WebRTC application will use this token to forward WebRTC requests to both STUN and TURN servers (Alan, 2014).

NON BROWSER APPLICATIONS

WebRTC is easy to interoperate with non-browser devices, too. There are multiple open source libraries which provide JavaScript SIP client. These libraries can be used to interact with PSTN switches and place phone calls to regular phone numbers. One good example of these libraries is jsSIP, sipML5 and Phono (Uberti & Dutton, n.d.).

WEBRTC AND EDUCATION TECHNOLOGY

Current live video conferencing solutions are either not designed for educational purposes, or proprietary protocols and software. Take for instance Microsoft Skype or Google Hangouts. There is no way to integrate Google Hangouts or Skype in educational applications. Students and teachers must sign up separately to use the service (Manson, 2013). That could be easy for some age groups. But with other students not old enough or who have restricted access to Internet websites for privacy and safety reasons, that could be very inconvenient. RTMP is a protocol owned by Adobe for real-time communication and it can be integrated with any application. But this comes with high fees which also make it not a good choice. In contrast, having video chat systems integrated with the educational system using a single sign on (in which the students and the teachers just sign on in only one system) can make all the difference.

CONCLUSION

WebRTC will revolutionize the live video conferencing and the real-time communication. The applications for this technology are endless. WebRTC, as an open source project, will allow any company to produce live video conferencing. And collaborative applications for educational purposes will make it easier to develop while also being cost effective. These kinds of applications used to cost a lot and involve complicated setup and infrastructure.

ACKNOWLEDGMENTS

I am grateful to Professor David Joyner for providing me with the opportunity to explore this great topic, and to my mentor Jamy Castro for her valuable feedback about my work.

REFERENCES

Alan, J. B. (2014). *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web (Kindle Location 3905)* (Kindle eBook Edition ed.). St. Louis: Digital Codex, LLC.

Manson, R. (2013). Real-time programming. In R. Manon, & S. Mogre, *Getting Started with WebRTC*. Birmingham, England: Packt Publishing. Retrieved from https://ebookcentral-proquest-com.prx.library.gatech.edu

Uberti, J., & Dutton, S. (n.d.). *http://io13webrtc.appspot.com/*.

Uberti, Justin; Thatcher, Peter;. (2018, April 22). *WebRTC*. Retrieved from wikipedia: https://en.wikipedia.org/wiki/WebRTC